NAVAL POSTGRADUATE SCHOOL
MONTEREY, CALIFORNIA

THESIS

DecisionNet -- A PROTOTYPE DISTRIBUTED DECISION SUPPORT SYSTEM SERVER

by

Andrew S. King

September 1995

Thesis Advisor: Hemant K. Bhargava

Approved for public release; distribution is unlimited.

19960328 003

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE September 1995	3. REPORT TYPE AND DATES COVERED Master's Thesis	
4. TITLE AND SUBTITLE: DecisionNet -- A PROTOTYPE DISTRIBUTED DECISION SUPPORT SYSTEM SERVER		5. FUNDING NUMBERS	
6. AUTHOR(S) Andrew S. King			
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey CA 93943-5000		8. PERFORMING ORGANIZATION REPORT NUMBER	
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)		10. SPONSORING/MONITORING AGENCY REPORT NUMBER	

11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release; distribution is unlimited.	12b. DISTRIBUTION CODE

13. ABSTRACT *(maximum 200 words)*

   This thesis documents the design and prototyping of DecisionNet -- a World Wide Web accessible decision support system server. Most decision support software is sold as a product. With DecisionNet, we attempt to shift this paradigm by providing decision support systems as a *service* vice a *product*.

   DecisionNet takes advantage of the hardware and software independence of the World Wide Web to provide connectivity between consumers, providers and DecisionNet. In the DecisionNet environment, the decision technology resides on and is executed by the provider's computer system. The consumer's data is set to the provider's computer via DecisionNet, allowing for anonymity of the consumer, automating the format conversions required for the decision technology, and minimizing the administration of user accounts for the provider.

14. SUBJECT TERMS Decision Support, Distributed Environment, Heterogeneous, Models, Algorithms, Prototype,	15. NUMBER OF PAGES 79
	16. PRICE CODE

17. SECURITY CLASSIFICA-TION OF REPORT Unclassified	18. SECURITY CLASSIFI-CATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICA-TION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT UL

DecisionNet -- A PROTOTYPE DISTRIBUTED DECISION SUPPORT SYSTEM SERVER

Andrew S. King
Lieutenant, United States Navy
B.S., Oregon State University, 1987

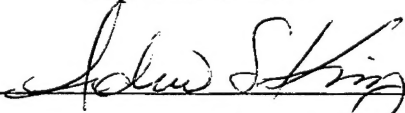Submitted in partial fulfillment
of the requirements for the degree of

MASTER OF SCIENCE IN INFORMATION TECHNOLOGY MANAGEMENT

from the

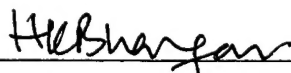**NAVAL POSTGRADUATE SCHOOL**
September 1995

Author: _____
Andrew S. King

Approved by: _____
Hemant Bhargava, Thesis Advisor

_____
Balasubramaniam Ramesh, Second Reader

_____
Reuben T. Harris, Chairman
Department of Systems Management

iii

iv

ABSTRACT

This thesis documents the design and prototyping of DecisionNet -- a World Wide Web accessible decision support system server. Most decision support software is sold as a product. With DecisionNet, we attempt to shift this paradigm by providing decision support systems as a *service* vice a *product*.

DecisionNet takes advantage of the hardware and software independence of the World Wide Web to provide connectivity between consumers, providers and DecisionNet. In the DecisionNet environment, the decision technology resides on and is executed by the provider's computer system. The consumer's data is set to the provider's computer via DecisionNet, allowing for anonymity of the consumer, automating the format conversions required for the decision technology, and minimizing the administration of user accounts for the provider.

TABLE OF CONTENTS

I. INTRODUCTION

This thesis describes the design and implementation of DecisionNet[1] -- a distributed decision technologies[2] server for the World Wide Web.[3] DecisionNet utilizes the Internet and the Hypertext Transfer Protocol (HTTP) of the World Wide Web for connectivity.

A. THE IDEA BEHIND DecisionNet

DecisionNet is based on the premise that decision technologies would be distributed wider and used more if treated as services rather than as products. The purchaser of a *product* -- such as an automobile or computer hardware -- takes possession of and completely controls the item. A purchaser of a service -- such as cable television or a rental car -- only *uses* the product, and pays the owner for that usage.

In the computer software market, the purchaser of a software product is responsible for having the correct hardware and software environment for the new software. A person would not purchase a Macintosh version of a program

---

[1] The initial development team for DecisionNet was Professor Hemant Bhargava, Captain Danny McQuay, USMC, and Lieutenant Andrew King, USN, all of NPS, and Professor Ramayya Krishnan, Carnegie Mellon University.

[2] Decision technologies refers to a variety of software used to support decision making and modeling for analysis of data.

[3] The following acronyms are used in this paper: HTML (hypertext markup language), DSS (decision support systems), OR (operations Research), WWW or the *Web* (World Wide Web), the *Net* (Internet).

1

to run on an IBM compatible computer. The hardware and software requirements are marked on each product available for purchase.

Under DecisionNet, software compatibility is no longer an issue, since the only software actually executed on the local computer is the WWW browser and Internet access software. Current mechanisms for distributing decision technology software[4] and compatibility issues are discussed in more detail in Chapter II.

B.   MOTIVATIONS FOR DecisionNet

What does DecisionNet offer that current mechanisms -- including the Internet -- for the development, distribution and use of decision technologies do not? Over the last few years, the Internet and WWW have allowed remote users to "look up" or "download" data, documents, and executable software. Rarely does this retrieval involve the real-time, and interactive specified, *mathematical processing* of data. UNIX based computer system users have been able to remotely execute programs via *Telnet* and *rsh*, however many new users are not on UNIX based systems. That, effectively, means that the expertise encoded in an OR/MS model or algorithm cannot be delivered in this fashion unless, of course, a copy of that model or algorithm is transferred; this requires the receiver to have a suitable hardware and software environment for working with that copy. More motivations are detailed in Chapter II.

---

[4] Software, Program, and Script are used interchangeably in this paper to refer to an executable computer program regardless of the language used or the method of execution -- interpreted, compiled or assembled.

C.   ENVIRONMENT FOR DecisionNet

There are three kinds of players in the DecisionNet environment --

consumers, and providers of decision technologies, and agents. Consumers have

decision problems to analyze. Providers have a decision technology or model

they wish to make available to the consumers. The DecisionNet server is a

collection of software agents that assist the interactions between the consumer

and provider. DecisionNet agents allow these interactions to take place over a

heterogeneous network with minimal requirements for both parties. Some of

the design factors for the DecisionNet environment are detailed in Chapter III.

D.   OUTLINE FOR REST OF THESIS

In Chapter II, I will discuss the current mechanisms for distributing

decision technologies. In Chapter III, I will discuss some of the design factors for

the DecisionNet environment . In Chapter IV, I will delve into the design of the

current DecisionNet software design and implementation including a designer's

notebook for the major software modules. Lastly, in Chapter V , I will

summarize the further research opportunities in the DecisionNet environment.

II. COSTS TO ACQUIRE DECISION TECHNOLOGIES

What are decision technologies? Decision Technologies are complex software packages that are made up of multiple parts. These parts could be one or more of the following -- a database management system, a model-base management system, a dialog generation and management system, and a user interface. [Ref. 1; Ref. 2, p. 382]

DSS may be divided into two categories -- the "institutional DSS" and the "quick-hit DSS". [Ref. 2] The "Institutional DSS" is generally built by Information Technology (IT) professionals and is intended to become a part of the organizational infrastructure. The "quick-hit DSS" is generally built by managers or end-users to help in making a decision. The decision may be a reoccurring decision or a one-time decision. The three sub-types of a "quick hit DSS" are a reporting DSS, short analysis programs, and those built with a DSS generator. [Ref. 2]

A. ACQUISITION OF AN INSTITUTIONAL DSS

Institutional DSS are defined as

... systems built [and maintained] by professionals, often decision support groups. These systems are intended for organizational support on a continuing basis, and they are generally written using a decision support language. [Ref. 2 p. 383]

Examples of institutional DSS include a marketing analysis DSS built for Ore-Ida Foods, Inc. [Ref. 2, p. 383] and a system designed and built by MicroStrategy for the Air Force to analyze over two hundred military, economic, and social impact characteristics for each base recommended for closure to the Base Closure and Realignment Commission (BRAC). [Ref. 4]

For a small business or individual without an IT staff, the development of an institutional DSS would be contracted out to a consultant group. This option is prohibitively expensive for most businesses, reducing the number of businesses that use DSS products for problem solving.

B. CREATING QUICK HIT DSS

1. Creating "Reporting" DSS
Reporting DSS are

> ... used to select, summarize, and list data from *existing data files* [italics mine] to meet managers' specific information needs. ... a few mathematical operations may be performed. If computer graphics are used, however, then trends, variances, and so on can be shown. [Ref. 2, p. 385]

These DSS are closely related to the Executive Information Systems (EIS) built for the highest levels of management, emphasizing collecting data from various sources, reporting with fast response, flexibility, and graphical presentation. The Reporting DSS is the most widely used, and will continue to be used, by managers who are required to analyze information stored in

computer systems that are both internal and external to the business or corporation. [Ref. 2, p. 385]

An example of a commonly used Reporting DSS for WWW servers is the WWW server statistics programs. The statistics program scans the access log file that is created by the server whenever an access is requested, and tabulates all of the entries. The WWW access log for the DecisionNet Welcome server is located at *http://dnet.as.nps.navy.mil/WebStat/machttp.html*.

2. Creating "Short Analysis Programs"

These programs analyze data as well as print or display the data ... Managers can write these short programs themselves, and they generally only use a small amount of data, which may be entered *manually* [italics mine] [Ref. 2, p. 385]

An example of this type of program is one that determines the payment schedule for a home mortgage at different interest rates. [Ref. 3] It is a simple program that is extremely useful in relating the cost of various mortgage options.

The short analysis DSS differs from the reporting DSS in the amount and location of the data being analyzed. The reporting DSS might have large amounts of data to scan through to find a few relevant data points, while the short program DSS uses a limited amount of data.

3. Building DSS with a DSS Generator

A DSS generator allows a user to develop specific DSS by modifying the modules provided with the generator program. The DSS modeling and database

components are modified to fit the new specific DSS and the user interface is created to best display the data for interaction with the user. [Ref. 1, p. 292]

4. Basic Spreadsheet DSS

Modern spreadsheet programs, such as Lotus 1-2-3 and Microsoft Excel provide extremely powerful analysis tools, allowing users to create simple DSS that include mathematical functions and linear programming.

> ... spreadsheets are extremely strong in dialog support. [user interface] They allow users to make corrections, additions, and deletions quickly and easily as well as perform numerous what-if analyses by changing some values and seeing the results. [Ref. 2, p. 389]

In addition to the what-if analysis support, spreadsheets have graphing modules that may display the results in a graphical form. Spreadsheets are intuitive for most users to use, since they display the data and relationships in a familiar table format and show the results instantly when a user changes variables in the table.

C. COMPARISON OF COSTS

The lifecycle[5] cost for the institutional DSS is high due to the IT staff that is required to create, operate and maintain the DSS. The Quick-Hit DSS is costly in a different manner. The person who created the DSS is likely to continue development of the DSS on company time. With the institutional DSS, the documentation is maintained within the organization, while with a quick-hit

---

[5] The lifecycle of a DSS refers to the total cost of the system from inception until the retirement of the system

DSS the documentation is usually not written. The person who built the quick-hit DSS is the only person who understands the inner workings it. If this individual leaves the organization, all working knowledge of the DSS leaves as well.

D.   CONCLUSIONS

Since institutional DSS are expensive in monetary terms, and quick hit DSS are expensive in personnel time, perhaps a new paradigm needs to be used -- *using a DSS service.* In many areas of the economy, the service industry thrives. Using a service is economically viable if the cost of using the service is less than the cost of purchasing the product.

To use a DSS service, a business would need to establish a reliable source of DSS services for a lower overall cost than owning a DSS product. The DSS service should provide secure analysis of the data provided by the consumer. The consumer should provide the data in a format acceptable to the DSS service and receive the results of the analysis requested. The DSS service should provide the widest selection of available DSS services for selection to the consumer, and conduct any data conversions required.

For the provider of a DSS service, there should be a means of advertising the analysis services available to the widest consumer base. Formatting of the consumer's data could be preprocessed by a brokerage prior to being sent to the

provider. The brokerage should perform the same function for the outputs from the provider.

DecisionNet is a prototype DSS service -- providing consumers and providers a DSS Service agency. DecisionNet provides the consumer with a "Yellow Pages" of available DSS services, data formatting, and the generated analysis outputs. DecisionNet delivers the providers of DSS services the widest advertising of their services.

When DecisionNet is fully operational, the providers and consumers would conduct business through DecisionNet's agents. Chapter III discusses the design of the DecisionNet environment in more detail.

III.   DESIGN OF THE DecisionNet ENVIRONMENT

During every design process, there are multiple decisions to be made. Each decision shapes the final solution or product. With a different decision early in the project, an alternative solution would be created. In this chapter, I will show some of the thought processes that were followed during the initial design of the DecisionNet environment.

Before designing DecisionNet, we made several assumptions. We assumed that the World Wide Web would continue to grow at the same tremendous rate that it has shown in since January 1993. [Ref. 5] The most accurate tracking records for the number of bytes transmitted have been maintained by the NSFNET since January 1988 until the decommissioning of the network in April 1995. Figure 1 shows the monthly World Wide Web traffic history for the NSFNET from December 1992 to April 1995. In "... December 1994, [traffic] numbers begin decreasing as traffic migrates to the new NSF architecture, for which no comparable statistics are available." [Ref. 6] To reach the largest possible number of consumer for DSS services, we decided to utilize the WWW for connectivity in the DecisionNet environment.

A second assumption we made before designing DecisionNet was that if we built a DSS service, both providers and consumers would use it. With the explosion of information available to the on-line community, good analysis tools need to be made available to assist in solving problems.

NSFNET Monthly World Wide Web Traffic History

(In Bytes Transmitted)
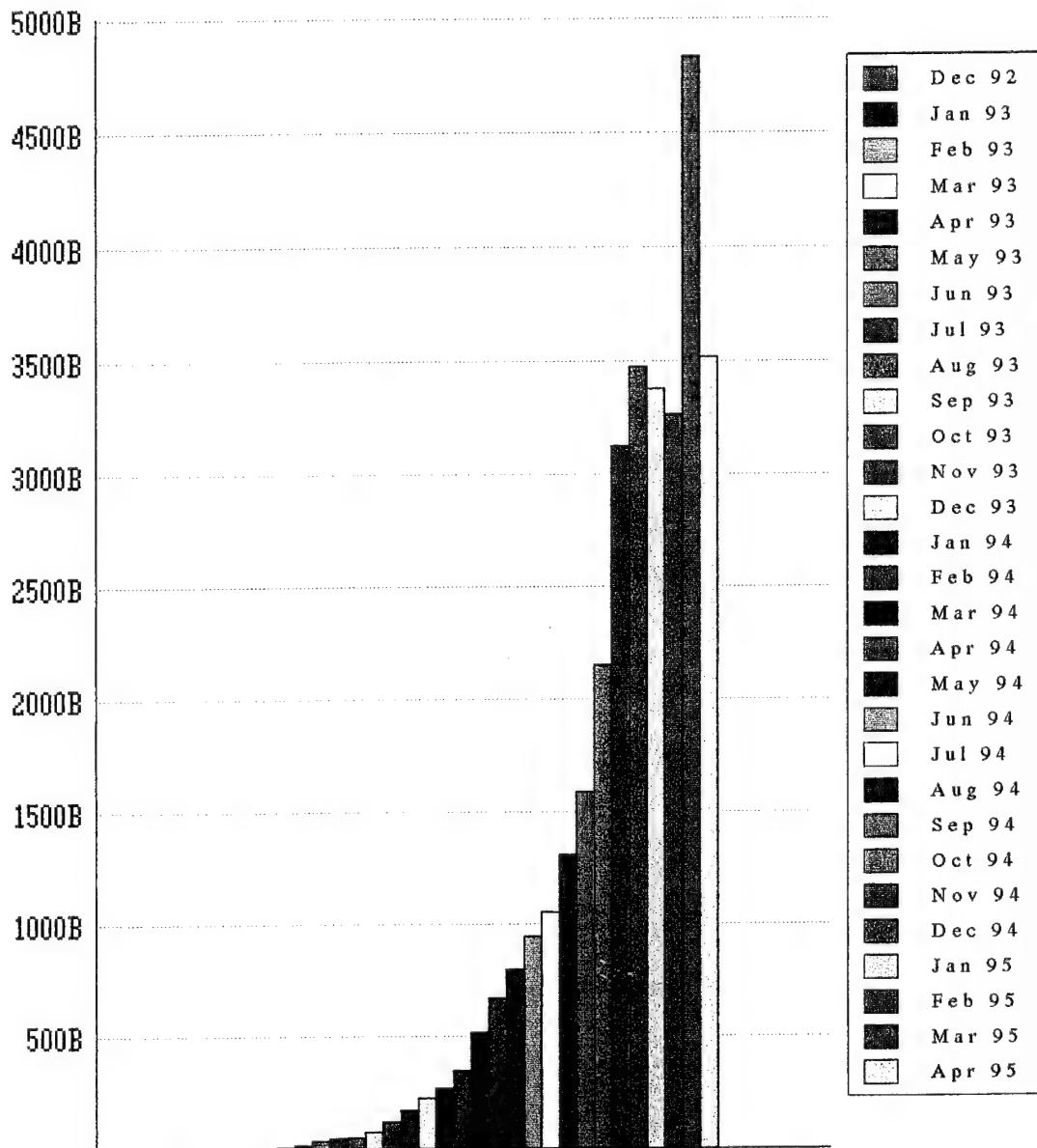
Figure 1: NSFNET Monthly World Wide Web Traffic History From
December 1992 to April 1995.   From Ref. [6]

A.   DESIGN OBJECTIVES

While designing DecisionNet, we had three objectives: to increase the availability of decision technologies for problem solving, to ensure the system designed requires only a minimal amount of human interaction to continue to operate, and to allow simple single algorithm problems as well as complex multiple algorithm problems to be solved across the Internet. The environment we designed for DecisionNet to operate within must contribute to these objectives.

By utilizing the WWW for connectivity, we eliminated many of the specific hardware requirements imposed by many software products. HTTP is a hardware independent protocol, allowing any person with a WWW connection to participate as a consumer of DSS via DecisionNet. This helps DecisionNet reach the first design objective -- increase availability of decision technologies for problem solving.

The second design objective, ensuring the system requires only a minimal amount of human interaction to continue to operate, may be reached through the use of software agents. These software agents can provide the functionality of a human for a specific task, such as account registration and searching data files. Maintenance of the HTML files is automated with exception reports sent to the DecisionNet development team via Electronic Mail (E-Mail).

For example, when a new technology is added to DecisionNet, the system automatically generates an E-Mail to the system administrator, and created a

new HTML for the technology. At a pre-determined time, during the time of lowest system utilization, a DecisionNet agent moves the HTML file into the indexable directories and re-indexes the directories. Through use of UNIX system functions such as *crontab* and *sendmail*, DecisionNet can automate many of the routine functions that normally require human interactions.

The third design objective is to allow simple *single algorithm* problems as well as complex *multiple algorithm* problems -- either sequential or parallel -- to be solved across the Internet. The concept is simple for single algorithm problems -- send the data to be analyzed to the remote site, log in to the remote site and execute a program to analyze the data set sent previously, capture the results and send them back to the remote user.

For sequential algorithmic problems, the results from one analysis are used as the input for another algorithm. This forces either the consumer or DecisionNet to convert the output format from one algorithm into the correct input format for the next algorithm. These conversions can be non-trivial, however intelligent agents may be developed to manage these conversions.

For parallel algorithmic problems, each parallel path could be treated like a sequential problem, until the parallel paths are united. More research in the area of intelligent agents which can control this parallel processing is needed.

B. DESIGN PARAMETERS

To achieve the design objectives, the design team defined an environment for DecisionNet to operate within. We defined who may communicate with whom, how do they communicate, and what protocols to use.

1. Who May Communicate With Whom

There are many different ways to design a communications infrastructure. I will describe two design possibilities for the communications structure of the DecisionNet environment.

One method is to design DecisionNet as a simple index, with pointers to executable DSS already existing on the WWW. The consumer would communicate with DecisionNet to view the index and then directly communicate with the selected provider. This would provide a minimum level of service to the consumers and providers in the environment. This approach is not technically challenging, and may be accomplished through a simple HTML homepage with hypertext links. Data conversions must be completed either by the consumer or the provider. The accuracy of the index would be limited to the last update of the HTML page.

This method requires that each consumer establish an individual account with each provider's service they desire to use. This places an enormous administrative burden on the providers. It requires each provider's system administrator to establish short term (or one use) accounts and to maintain

security of their server. Despite all these limitations, the initial prototype of DecisionNet was a simple set of linked HTML pages.

A second method is to make DecisionNet the center of all communications. Figure 2 shows the DecisionNet centered communications network. This method allows for data conversions, formatting of inputs and outputs, monitoring of usage, and anonymity of the consumer and provider.

With this communications approach, the consumers obtain access to all decision technologies simply via a WWW browser. For larger data sets, the consumer would be required to have a FTP client to send the data files to DecisionNet prior to execution of the decision technology.

The aforementioned problems with establishing short term accounts at each server would be eliminated. Each consumer would establish an account with DecisionNet which authorizes them to utilize any DSS listed on the index. This is the method chosen for the current and future DecisionNet prototypes.

2. Communication Paths

Figure 2 shows a simplified communication path and protocols for a WWW accessible decision technology in a DecisionNet centered environment. In this example, it is assumed that DecisionNet has set up a consumer account. The provider must have registered their technology with DecisionNet and created an account for DecisionNet's use. This account is used by DecisionNet agents only, acting for a consumer. The data required by the decision technology is collected from the consumer and sent to the provider using the

DecisionNet account. The results are sent back to DecisionNet, which reformats the data for presentation to the consumer as previously requested.

In this model, the consumer account problem is shifted away from each provider and centralized with DecisionNet. The provider's administrative burden is reduced to a single account for DecisionNet vice the multitude of short term accounts for each user. DecisionNet assumes the responsibility for maintaining the large number of short term accounts. DecisionNet also can perform input and output conversions required between the provider and consumer.
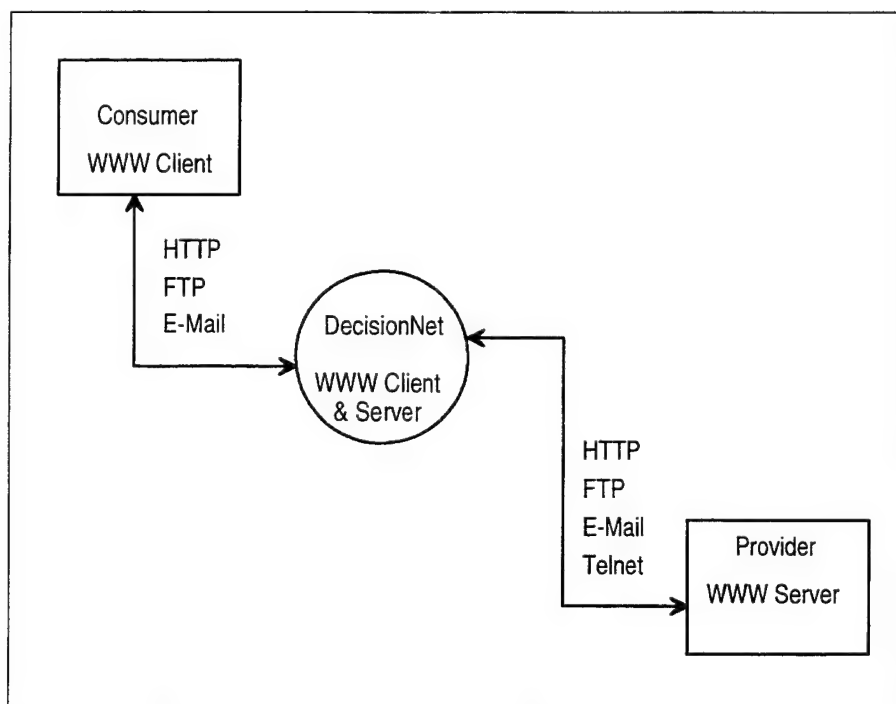
Figure 2: Communication Paths in The DecisionNet
          Environment

3. Communications Protocol selection

In creating the DecisionNet environment, there are numerous choices for communications protocols. The basic choices involved the use of a standard protocol or a specialized protocol.

The specialized protocol could be designed to optimize the functionality of the DSS and minimize the number of conversions required by any party. A specialized protocol requires either a specialized browser or a new Multipurpose Internet Mail Extensions (MIME) type with the required executable for the specific computer being used by the consumer. [Ref. 8] An example of using a MIME type is to have Word Perfect execute when ever a *.wp* file is downloaded. Using a specialized protocol would restrict the number of providers and consumers who would be able to use DecisionNet. Further research based on this approach is currently in progress at Carnegie Mellon University (USA), Monash University (Australia), Institut Fur Wirtschaftsinformatik (Germany) and Arizona State University (USA). [Ref. 9; Ref. 10]

A standardized protocol, in particular HTTP and FTP, already has a large user base on the Internet. The entire WWW is based on HTTP, and the Internet is based on the TCP/IP protocol, which includes FTP. The current WWW browsers, such as NSCA's Mosaic and Netscape Navigator, support HTTP and the TCP/IP protocols from within the browser. This allows every WWW user to be a potential consumer for DecisionNet.

Each has its advantages, but our overriding concern is to maximize the available consumers and providers. To meet this objective, we chose to use a standard HTTP protocol for the basic DecisionNet environment.

C.   STATUS OF DecisionNet

We have defined two types of decision technologies that are executable on the WWW. An independent technology is one that has been setup by a provider to execute on the WWW. Providers of such technologies register their technology with DecisionNet, and require no other action from DecisionNet to execute their technology. At the current time, DecisionNet accepts all independent technologies.

The dependent technology requires DecisionNet to process input and output conversions for the consumer. Either the provider's technology is not designed for execution via the WWW, or it may require a non-anonymous *Telnet* session be established with their computer to execute the technology. This requires the DecisionNet server to open sockets in the background to send the information to the provider's server, while processing the information from the consumer. At the current time, DecisionNet is not capable of accepting registration of dependent technologies.

IV. SOFTWARE DESIGN AND IMPLEMENTATION

A. MODULAR DESIGN

The major functions of DecisionNet are Login, Registration of consumers and providers, Registration of Decision Technologies, Search, and Execution. Each major function is designed as a separate module. The modular design allows for easier updates and modifications to a single function without affecting the rest of DecisionNet. All modules use a set of Perl[6] libraries -- *cgi-lib.pl* and *dnetlib1.pl* -- for common functions such as checking the last modification time on a file and verification of a user name and password.

Appendix A contains functionality tables, cross referencing the function to the HTML file, data file(s) and Perl programs used. An annotated source code listing of *cgi-lib.pl* is in Appendix B, *dnetlib1.pl* in Appendix C. Detailed information for the Login module is in Appendix D, User Registration in Appendix E, and Technology Registration in Appendix F.

---

[6] "Perl" is an acronym for "Practical Extraction and Report Language," although Larry [Wall] has been known to claim that it really stands for "Pathologically Eclectic Rubbish Lister." [Ref. 11]

B. WHAT PROGRAMMING LANGUAGE TO USE

The processing of *forms* requires a program or script to receive and process the data at the WWW server. On the UNIX platforms, the de facto standard for a programming language for processing *forms* is Perl. The PC based platforms have a variety of scripting languages to process *forms* including Visual Basic and Applescript. During the initial development of DecisionNet, Perl was not available for the PC based servers.

In choosing the programming language to use for *forms* processing, one major consideration was the hardware executing the scripts. The PC platforms are limited in the number of simultaneous connections -- the IBM systems are limited to 16 [Ref. 12] and the Macintosh platforms are limited to 48 connections including all Telnet, FTP and WWW connections. [Ref. 13] The UNIX platforms have a much larger capacity for simultaneous connections.

There are a large number of WWW server scripts written in Perl that are in the public domain and available for modification to fit DecisionNet. Again, during the initial development of DecisionNet, there were few scripts available for the PC based servers.

In light of the limitations of the PC based servers, and the availability of UNIX WWW server(s) at the Naval Postgraduate School, we decided to base the *forms* processing portion of DecisionNet on the UNIX platform.

C.   FULL TEXT SEARCH VICE RELATIONAL DATABASE

DecisionNet is more than just a simple "yellow pages" that could be satisfied by a set of HTML pages. The dynamic nature of DecisionNet, with the ability for providers to add their technologies to the index, some technique for indexing the inputs needed to be established. Two of the methods currently used on the WWW are full text searches of documents and database queries.

Full text search engines, such as Lycos [Ref. 14], Harvest [Ref. 15] and SWISH [Ref. 16], index directory trees on a WWW server by document. Currently all three of these search engines are only available for UNIX based servers. If a word being searched for occurs in a document, the search results will return the entire document, but not to the specific line where that word was used.

Relational databases, such as Oracle or Sybase, store the information into separate tables that allow a specific line of information to be retrieved. The fields being retrieved are selectable by the query. The problem with relational databases is the lack of a standard interface with the WWW via Structured Query Language (SQL). Future research will examine how to execute searches using SQL and display the results for the consumer via the WWW.

We implemented a SWISH search engine for indexing DecisionNet, allowing full context searches of the registered decision technologies. To avoid the problems inherent with a document based search, we save information on each technology in a separate HTML file. This allows the SWISH search engine

to return a single decision technology as the results of a full text search for a string.

D.   PLATFORM FOR DecisionNet SERVER(S)

Each of the distinct functions -- Login, Registration, Search, Execution -- may be performed on multiple servers. As detailed in the last chapter, *forms* processing will be performed on a UNIX based system.

One reason for splitting the modules among multiple servers is that in the case of extremely heavy traffic, a single server might be overwhelmed by requests. Appendix A shows the overlap between the various modules and the data files. Due to these overlaps, we based DecisionNet on a Sun SparcStation 10 for everything except the static HTML files. All of the scripts, data files, access records, the Harvest search engine, and HTML files for dynamically generated HTML pages are on the SparcStation.

The static HTML files were placed on a PowerMac 7100 WWW server. The static files are a simple "Welcome" and "about DecisionNet" HTML pages, which will be accessed at a significantly lower rate than the main server. This network design is also used as a proof of concept that the DecisionNet servers can be separated and still function as a single unit to the outside world.

E.   CREATING A STATEFUL HTTP

HTTP is a stateless protocol, which " ... runs over a TCP connection that is held only for the duration of one operation." [Ref. 17; Ref. 18] Each request from the client to the server is a separate request, including graphics embedded on a single HTML page.   A portion of the *sm.nps.navy.mil* access log is reproduced below, showing a request for a HTML document, *dNetmenu.html*, which contains two graphics, *home_btn.gif* and *go_top_b.gif*, from a user in Japan. The first document requested is *dNetmenu.html*, followed rapidly by two requests for the embedded graphics.

```
pross120.u-aizu.ac.jp - - [12/Aug/1995:03:47:49 -0700] "GET /dnet/dNetmenu.html
pross120.u-aizu.ac.jp - - [12/Aug/1995:03:47:51 -0700] "GET /dnet/Gifs/home_btn.gif
pross120.u-aizu.ac.jp - - [12/Aug/1995:03:47:51 -0700] "GET /dnet/Gifs/go_top_b.gif
[Ref. 19]
```

To maintain the account information between separate HTML requests for DecisionNet, there are several options. One option requires the server to maintain an Access Control List (ACL) for each directory of files to be accessed [Ref. 20], or to embed hidden fields into the HTML document requested. The first option requires the Webmaster of the WWW server to establish an ACL for each directory to be accessed. In a dynamic environment such as DecisionNet, the ACL would be changing many times, requiring modifications to the ACLs to be done by script or by human intervention. This creates a possible security problem for the rest of the WWW server.

The second option requires each script to check the password files each time the script is executed. Buttons are presented to the user for each action.

Attached to each button, using hidden fields, is the user type, name and password. When the button is pushed, causing the requested action to be sent to the DecisionNet server, the hidden fields are sent with the data entered by the user. This requires each script to verify the user name and password prior to executing the remainder of the request. This process slows down the processing of the primary request and requires the access data files be checked for each request.

When designing DecisionNet, the ACLs for the original server processing the *forms* were not accessible to the DecisionNet Programmers. In light of this, we decided to maintain separate DecisionNet access files. This also reduces the turn around time required when adding new users into the DecisionNet access files.

F.   STATUS OF DecisionNet DEVELOPMENT

The current working prototype for DecisionNet is located at *http://dnet.as.nps.navy.mil.* The Consumer and Provider Registration, Login, and Technology Registration are all functional through the use of Perl scripts. The Search function uses a WAIS front end for the SWISH search engine. [Ref. 16] The data provided by the providers in the initial registration of a decision technology is stored in a temporary directory, and moved into correct directory manually. The server must be re-indexed to include the new technology.

The modification and deletion of decision technologies is currently done manually as well. The provider would send an E-Mail to request a deletion of the technology. The system administrator would take the E-Mail, delete the technology file, and re-index the server. All new accounts and new technologies are verified via E-Mail to the consumer or provider and a copy is sent to DecisionNet.

V. CONCLUSIONS AND FURTHER RESEARCH

This thesis accomplished the design and development of a prototype distributed decision support system. By utilizing the World Wide Web for connectivity, we have eliminated many of the hardware and software restrictions that consumers have when attempting to use DSS services. The current prototype is a modular framework which allows for future expansion and continual improvement.

A. FURTHER RESEARCH TOPICS

The basic structure for DecisionNet has been established and is functional. The following is a listing of areas where DecisionNet could be improved:

1. How should background process be utilized to *telnet* into a remote site and perform transactions remotely.

2. How should raw data input from the consumer be processed into formatted data usable by a functional model.

3. How should the collection of complex input sets be automated.

4. What set of taxonomies should be used for cataloging the decision technologies.

5. Conversion from an HTML based search engine into a relational database, allowing greater flexibility for searching and indexing the decision technologies.

6. Programming of Intelligent agents for controlling parallel processing of complex multiple algorithmic problems

The continued development of DecisionNet requires background processing to connect with the provider and execute programs. Lacking these developments restricts DecisionNet to processing independent technologies only.

B. CONCLUSIONS

The prototype for DecisionNet is currently functional and on line at *http://dnet.as.nps.navy.mil/dnethome.html.* Currently the functionality is minimal, since DecisionNet only allows independent technologies to be accessed. The concept of sending the data to be processed at a remote site is not a novel idea, however our approach, using the World Wide Web for connectivity, is novel. The DecisionNet design is a viable approach to placing executables which require user data on the World Wide Web. In the future, one should expect to see more implementations of the DecisionNet concept in areas unrelated to decision support systems.

APPENDIX A.   FUNCTIONALITY TABLES

The first table cross references all functions of DecisionNet with the Perl programs, and HTML files used. There are two types of HTML files that are used in DecisionNet -- Static and Dynamic. The static files are complete HTML files, while the dynamic files are split into three sections -- headers, body and footers. The header files include the title and header information of a HTML document. The body section contains the body of the document and the footer contains the ending of the HTML document, including the closing </HTML> tags. The *.html* files in the main DecisionNet directory or in the html directory on *sm*. The *.body* files are stored in the body directory.

The primary purpose for using dynamically generated HTML documents is to allow the insertion of hidden fields into the document "on the fly." The hidden fields are usually the usertype, used to determine the file to use for the access datafile, username and password. The password is encrypted using a UNIX system function, *crypt*, which uses the data encryption standard (DES). The Perl scripts used in the generation of the HTML documents retrieve the document pieces to be sent to the client in the order needed, parsing the correct information into the hidden fields as required. All *.pl* files are Perl scripts stored in the Scripts directory on the *sm* server.

	Login of all Users	Registration of all Users	Registration of Decision Technologies
Perl Programs Used	*login.pl*	*register.pl*	*Register.Form.pl* *add.pl*
HTML Page	dNetmenu.html	register.html	N/A
HTML Body	Prov_menu.body User_menu.body	N/A	Model.Register.body

Table 1: Cross Reference of Function to HTML And Perl Files Used

Each of the datafiles and Perl libraries is used by several of the functions.

The following table cross references the function to the datafile and library used.

	Login of all Users	Registration of all Users	Registration of Decision Technologies
dnetlib1.pl	X	X	X
cgi-lib.pl	X	X	X
Prov_Address.txt		X	X
Prov_Access.txt	X	X	X
Prov_mail.txt		X	
Prov.access.log	X		
User_Address.txt		X	
User_Access.txt	X	X	
User_mail.txt		X	
User.access.log	X		

Table 2: Cross Reference of Function to Data Files And Perl Libraries Used

APPENDIX B.   *CGI-LIB.PL*

This is a library of Perl routines to perform the Common Gateway Interface

(CGI) manipulations. DecisionNet uses the ReadParse function for every user input to

populate the associative array with the variables and data received. This library was

downloaded from Meng Wong's WWW pages at

http://www.seas.upenn.edu/~mengwong/forms/.

```perl
#!/usr/local/bin/perl -- -*- C -*-

# Perl Routines to Manipulate CGI input
# S.E.Brenner@bioc.cam.ac.uk
# $Header: /cys/people/seb1005/http/cgi-bin/RCS/cgi-lib.pl,v 1.7 1994/11/04
00:17:17 seb1005 Exp $
#
# Copyright 1994 Steven E. Brenner
# Unpublished work.
# Permission granted to use and modify this library so long as the
# copyright above is maintained, modifications are documented, and
# credit is given for any use of the library.
#
# Thanks are due to many people for reporting bugs and suggestions
# especially Meng Weng Wong, Maki Watanabe, Bo Frese Rasmussen,
# Andrew Dalke, Mark-Jason Dominus and Dave Dittrich.

# see http://www.seas.upenn.edu/~mengwong/forms/   or
#    http://www.bio.cam.ac.uk/web/              for more information

# Minimalist http form and script (http://www.bio.cam.ac.uk/web/minimal.cgi):
# if (&MethGet) {
#   print &PrintHeader,
#     '<form method=POST><input type="submit">Data: <input name="myfield">';
# } else {
#   &ReadParse(*input);
#   print &PrintHeader, &PrintVariables(%input);
# }
```

```perl
# MethGet
# Return true if this cgi call was using the GET request, false otherwise
# Now that cgi scripts can be put in the normal file space, it is useful
# to combine both the form and the script in one place with GET used to
# retrieve the form, and POST used to get the result.

sub MethGet {
  return ($ENV{'REQUEST_METHOD'} eq "GET");
}


# ReadParse
# Reads in GET or POST data, converts it to unescaped text, and puts
# one key=value in each member of the list "@in"
# Also creates key/value pairs in %in, using '\0' to separate multiple
# selections

# If a variable-glob parameter (e.g., *cgi_input) is passed to ReadParse,
# information is stored there, rather than in $in, @in, and %in.

sub ReadParse {
  local (*in) = @_ if @_;


  local ($i, $loc, $key, $val);

  # Read in text
  if ($ENV{'REQUEST_METHOD'} eq "GET") {
   $in = $ENV{'QUERY_STRING'};
  } elsif ($ENV{'REQUEST_METHOD'} eq "POST") {
   read(STDIN,$in,$ENV{'CONTENT_LENGTH'});
  }

  @in = split(/&/,$in);

  foreach $i (0 .. $#in) {
   # Convert plus's to spaces
   $in[$i] =~ s/\+/ /g;

   # Split into key and value.
   ($key, $val) = split(/=/,$in[$i],2); # splits on the first =.

   # Convert %XX from hex numbers to alphanumeric
   $key =~ s/%(..)/pack("c",hex($1))/ge;
```

34

```perl
    $val =~ s/%(..)/pack("c",hex($1))/ge;

    # Associate key and value
    $in{$key} .= "\0" if (defined($in{$key})); # \0 is the multiple separator
    $in{$key} .= $val;

  }

  return 1; # just for fun
}


# PrintHeader
# Returns the magic line which tells WWW that we're an HTML document

sub PrintHeader {
  return "Content-type: text/html\n\n";
}


# PrintVariables
# Nicely formats variables in an associative array passed as a parameter
# And returns the HTML string.

sub PrintVariables {
  local (%in) = @_;
  local ($old, $out, $output);
  $old = $*;  $* =1;
  $output .=  "<DL COMPACT>";
  foreach $key (sort keys(%in)) {
    foreach (split("\0", $in{$key})) {
      ($out = $_) =~ s/\n/<BR>/g;
      $output .=  "<DT><B>$key</B><DD><I>$out</I><BR>";
    }
  }
  $output .=  "</DL>";
  $* = $old;

  return $output;
}


# PrintVariablesShort
# Nicely formats variables in an associative array passed as a parameter
# Using one line per pair (unless value is multiline)
# And returns the HTML string.
```

```perl
sub PrintVariablesShort {
 local (%in) = @_;
 local ($old, $out, $output);
 $old = $*;  $* =1;
 foreach $key (sort keys(%in)) {
  foreach (split("\0", $in{$key})) {
    ($out = $_) =~ s/\n/<BR>/g;
    $output .= "<B>$key</B> is <I>$out</I><BR>";
  }
 }
 $* = $old;

 return $output;
}

1; #return true
```

APPENDIX C. *DNETLIB1.PL*

This set of Perl routines are unique to DecisionNet, but common to all of the modules. DecisionNet uses these routines to perform functions such as getting the current system time and date, checking the access files for a registered consumer or provider, and sending E-Mail from DecisionNet. Many of these routines were written with the assistance of Larry Wall's and Randal Schwartz's book, *Programming perl*[7].

```
#! /usr/bin/perl
#
#   DecisionNet Library
#
#  Written by Andrew King
# These are common routines for DecisionNet modules
# First Used: Feb 95
# Last Modified: Aug 95


# ----------------------------
# Sub DateTime
# Get and format current Date and Time
#  Returns formatted Date and Time
# ---------------------------

sub DATETIME {

  ($sec,$min,$hour,$mday,$mon,$year,$wday,$yday,$isdst) = localtime(time);
  $thisday = (Sunday,Monday,Tuesday,Wednesday,Thursday,Friday,Saturday)[$wday];
  $thismonth = (Jan,Feb,Mar,Apr,May,Jun,Jul,Aug,Sep,Oct,Nov,Dec)[$mon];
  $AccessDate = "$hour:$min:$sec $thisday $mday $thismonth 19$year";

  return $AccessDate;
}
```

---

[7] Wall, Larry and Randal L. Schwartz, *Programming perl* (Sebastopol, CA : O'Reilly & Associates, Inc, 1992)

37

```perl
#
# ---------------------------------------------------------
# Check Duplicate : Given a File Name, Username, Encrypted Password, and divider
# Check Duplicate returns either duplicate (1) or unique (0) to the
# calling program.
# ---------------------------------------------------------

sub CHECK_DUP {
  local ($File, $Usrname, $Pwd, $Div) = @_;
  $Dup = 'Unique';

  open (Input, $File) | | die "Can't open $File: $!\n";
  while (<Input>) {
   chop;
   ($User, $Password) = split(/$Div/, $_);
   if ($User eq $Usrname) {
     $Dup = 'Duplicate';
   }
  last if ($Dup eq 'Duplicate');
  }
  close ($File);
  return $Dup;
} # End of CHECK_DUP


# -----------------------------
# Sub Accesslog
# Prep and send login data to access log file : name and date
# ---------------------------

sub ACCESSLOG {
  local($File, $UserName) = @_;

  $AccessDate = &DATETIME;

  # >> opens file in append mode
  open (OutPutFile, ">>$File");

  print OutPutFile "$UserName,$AccessDate\n";

  close OutPutFile;
} # End Accesslog

#
# ---------------------------------------------------------
```

```perl
# Check Last Access : Given a File Name, Username, Encrypted Password, and divider
# Check LastAccess returns either No Access (1) or Access Date (0) to the
# calling program.
# -----------------------------------------------------------

sub LAST_ACCESS {
  local ($File, $Usrname) = @_;
$LastAccess = "Never";

  open (Input, $File) | | die "Can't open $File: $!\n";
  while (<Input>) {
   chop;
   ($User, $Access) = split(/,/, $_);

# Debug line
#  print "$User, $Access, $LastAccess<br>\n";

   if ($User eq $Usrname) {
    $LastAccess = $Access;
   }
  }
  close ($File);
  return $LastAccess;

} # End of LAST_ACCESS


# -----------------------------------------------------------
# Validate Account: Validate account receives File, Username,
# and Password and returns Valid (1) or invalid (0) to the
# calling program
# -----------------------------------------------------------

sub VALID {
 local ($File, $Usrname, $Pwd, $Div) = @_;
 $Valid = 'Invalid';
 open (Input, $File) | | die "Can't open $File: $!\n";
 while (<Input>) {
  chop;
  ($User, $Password) = split(/$Div/, $_);
# Debugging lines
# print "User Name from file: $User, From Form: $Usrname \n";
# print "Password from File: $Password, From Form: $Pwd\n";
  if ($User eq $Usrname) {
   if ($Password eq $Pwd) {
```

```perl
      $Valid = 'Valid';
     }
    }
   last if ($Valid eq 'Valid');
   } #End While

   return $Valid;

} # End of VALID

# --------------------------------------------------------
# Get POC Info from the datafiles
#
# --------------------------------------------------------
sub GetPOCInfo {
 local (*data) = @_;
 $Found = 'No';

 $InputFile = $data{'BaseFile'}.'data/'.$data{'UserType'}.'_Address.txt';

# opens file in read mode

 open (INPUT_FILE, $InputFile);
# Write out the address to THE_FILE
# Process each line in input file

 while (<INPUT_FILE>) {

   chop;  # Remove newline from end of input string

# Retreive the username and password from the password file
   ($UserName, $Company, $RealName, $EMail, $Street, $City, $State, $Zip, $Country)
= split(/$data{'Comma'}/, $_);

   $UserName =~ y/\xfb\xfc\xfd//d;


# Check for matching username and Password
   if ($UserName eq $data{'Usrname'}) {
    $RealName =~ y/\xfb\xfc\xfd//d;
    $EMail =~ y/\xfb\xfc\xfd//d;
    $Street =~ y/\xfb\xfc\xfd/[^\xfb\xfc\xfd]/d;
    $Company =~ y/\xfb\xfc\xfd/[^\xfb\xfc\xfd]/d;
    $City =~ y/\xfb\xfc\xfd//d;
    $State =~ y/\xfb\xfc\xfd//d;
```

40

```perl
    $Zip =~ y/\xfb\xfc\xfd//d;
    $Country =~ y/\xfb\xfc\xfd//d;
    $Found = 'Yes';
    }
  last if ($Found eq 'Yes');
  } # End of Loop


  $data{'POCName'} = $RealName;
  $data{'POCEMail'} = $EMail;
  $data{'POCCompany'} = $Company;
  $data{'POCStreet'} = $Street;
  $data{'POCCity'} = $City;
  $data{'POCState'} = $State;
  $data{'POCZip'} = $Zip;
  $data{'POCCountry'} = $Country;



} # End GetPOCInfo


# ----------------------------------------------------------
# Send File from Disk  Header and Footer
# ----------------------------------------------------------

sub HEADER {
 local($File, $Title, $Switch) = @_;

 if ($Switch eq 'Header') {
  print "Content-type: text/html\n\n"; #inform server what's coming
 }

 open (FILE1, $File) | | die "Can't open $File: $!\n";
 while (<FILE1>) {
  chop;
  print $_;
  if (/<TITLE>/) {
   if ($Switch eq 'Header') {
    print $Title;
   }
  }
  elsif (/<H1>/) {
   if ($Switch eq 'Header') {
    print $Title;
```

41

```perl
      }
    }

   print "\n";
  }
  close (FILE1);
  }


###############################
# CheckFile Subroutine
# Opens currentaccess files and compares
# UserName and PassWord to names on File
# All Usernames and Passwords are stored in an encrypted format
# This subroutine encrypts the input to match with the stored name
###############################


sub CHECKFILE {  # Check the new user name against the current users
  local(*FORM) = @_;

  $InputFile = $FORM{'BaseFile'}.'data/'.$FORM{'UserType'}.'_Access.txt';


# Load Initial Value for Form{Dup}
  $FORM{'Dup'} = 'No';
  $FORM{'DupName'} = 'No';

  open (INPUT_FILE, $InputFile);

# Process each line in input file
  while (<INPUT_FILE>) {

    chop;   # Remove newline from end of input string

# Retreive the username and password from the password file
    ($UserName, $PassWord) = split(/$FORM{'Comma'}/, $_);

    $UserName =~ y/\xfb\xfc\xfd//d;
    $PassWord =~ y/\xfb\xfc\xfd//d;

#encrypt the Received UserName and PassWord to compare with file
#Encryption salt changed to protect passwords
 #   $UsrNm = crypt($FORM{'Usrname'},SALT);
    $UsrNm = $FORM{'Usrname'};
```

```perl
#   $Pass = crypt($FORM{'Pwd'},SALT);
    $Pass = $FORM{'Pwd'};

# debugging line
# print "$UsrNm, $UserName<br>/n";

# Check for matching username and Password
  if ($UserName eq $UsrNm) {
    $FORM{'DupName'} = 'Yes';
    if ($PassWord eq $Pass) {
      $FORM{'Dup'} = 'Yes';
    }
  }
  last if ($FORM{'Dup'} eq 'Yes');
} # End of Loop

close INPUT_FILE;


} #End Checkfile



# -------------------------------------------------------
# Send E-Mail Verification : Send account information verification
# to the user and dnet via E-Mail
# Input: E-Mail Address, UserName
# Output: Email to User And Dnet
# Return: Nothing
# Based on Mailto.pl by Doug Stevenson, doug+@osu.edu
# -------------------------------------------------------

sub EMAIL {
 local($EMail, $Usrname, $Subject, $Body) = @_;

# E-Mail for account verivication for DecisionNet
 $dnet = "dnet@sm.nps.navy.mil";

# Location of sendmail
 $sendmail = "/usr/lib/sendmail -t -n";

# fork over the mail to sendmail and be done with it

 open(MAIL,"| $sendmail");
```

```
# Cc to DecisionNet

  print MAIL <<EOM;
From $dNet
Cc: $dnet
From: <$dnet>
To: $EMail
Reply-To: $dnet
Errors-To: $dnet
Sender:<$dnet>
Subject: $Subject
X-Mail-Gateway: DecisionNet
X-Real-Host-From: $dnet
$Body

If there are any problems, please contact $dnet

EOM

  close(MAIL);

} #End of E-Mail

1; #return true for Library Routine
```

APPENDIX D.   LOGIN

The Login module is called from the main DecisionNet menu, dNetmenu.html, when a consumer or Provider requests access to DecisionNet. The radio button on the form provides the variable *UserType* to *login.pl*. This allows the program to access the correct access records. By splitting the provider's and consumer's access records, it is possible for two users, one a provider and one consumer, to have the same login name.

After the user has been verified as a registered user, the access log is checked and updated. *Login.pl* then sends the results back to the client, giving them the correct menu -- either the Consumer or Provider Menu. The HTML form that is sent back is "generated" by using a header file, a body file, and a footer file. This ensures that all documents look the same at the top and bottom (return buttons etc.) and allows the body to be parsed to insert a few variables before all *form* submission buttons.

This is the third or fourth generation login script. The newer sections pass only the variables that are needed by the sub-routines. The older sub-routines pass pointers to the entire array of variables that was received from the *forms* submitted.

The data files and HTML files that are used by *login.pl* are listed in

Appendix A.

```perl
#! /usr/bin/perl
#       Function: Login
#      Written by: Andrew King
#          Date: 23 Apr 95
# Parsing Using cgi-lib routines
#    Last Modified: 10 Aug 95
#         Version: 2.5
#         Purpose: Login Routine for DecisionNet

require 'dnetlib1.pl';
require 'cgi-lib.pl';


# -------------------------------
# Output UserType Menu
# -------------------------------

sub OUTPUT {  # Output the correct menu based on UserType from a file

 local(*Input, $AccessData) = @_;
 $HeaderFile = $Input{'BaseFile'}.'Head/header'.$Input{'Headnum'}.'.head';

# Select correct Header -- Else reject

 if ($Input{'UserType'} eq 'Prov') {
  $Title = "Provider Menu for ".$Input{'Usrname'}."\n";
 }
 elsif ($Input{'UserType'} eq 'User') {
  $Title = "Consumer Menu for ".$Input{'Usrname'}."\n";
 }
 else {  # Bogus Entry for User Type
  $Reject = 'Invalid Entry for User Type';
  &Reject($Reject);
 }

&HEADER($HeaderFile, $Title, 'Header');

$InputFile = $Input{'BaseFile'}.'Body/'.$Input{'UserType'}.'-menu.body';

stat($InputFile);
```

```perl
  $Age = -M _;
  $Age = int($Age);

print "<b>Your last access was at $AccessData<b>\n";


  open (File,$InputFile) || die "Can't open $InputFile: $!\n";

  while (<File>) {

    chop;
    $Line = $_;

    if (/TYPE=submit/) {
#   Insert Usertype, Username, and Encoded password before each button.

      print '<input name=UserType type=hidden value="',$Input{'UserType'},'">'."\n";
      print '<input name=Usrname type=hidden value="',$Input{'Usrname'},'">'."\n";
      print '<input name=Pwd type=hidden value="',$Input{'Pwd'},'">'."\n";
      print "$Line\n";
    }
    else {
      print "$Line\n";
    }

  }   # End While loop

  close (InputFile);


# print "<br><i>This file was last modified $Age Days ago</i><br>\n";

  $Title = "";
  $OutputFile = $Input{'BaseFile'}.'Foot/footer'.$Input{'Headnum'}.'.foot';
  &HEADER($OutputFile, $Title, 'Footer');

}   # End Output

# ----------------------------------
# Subroutine Reject
# ----------------------------------

sub REJECT {
  local ($Reject) = @_;
```

```perl
$HeaderFile = $Input{'BaseFile'}.'Head/header1.head';
$Title = "$Reject\n";
&HEADER($HeaderFile, $Title, 'Header');


print "<H1>Rejected !!! </H1>\n";

$Title = "";
$OutputFile = $Input{'BaseFile'}.'Foot/footer1.foot';
&HEADER($OutputFile, $Title, 'Footer');


} # End Reject


sub WHERETOGO { #Based on User Type Send different Printout

  local($UserType) = @_;

  if ($UserType eq 'Unreg') {
   $HeaderNumber = 'Bogus';
  }
  elsif ($UserType eq 'User') {
   $HeaderNumber = 2;

  }
  elsif ($UserType eq 'Prov') {
   $HeaderNumber = 3;
  }

  else {
   $HeaderNumber = 'Bogus';
  }
  return $HeaderNumber;
}

# Following deleted when included in dnetlib1.pl
# left here in case all else fails!
#
#
# -------------------------------
# Sub Accesslog
# Prep and send login data to file : name and date
# --------------------------
```

```perl
#
#sub ACCESSLOG {
#  local($File, $UserName) = @_;
#
# $AccessDate = &DATETIME;
#
# # >> opens file in append mode
# open (OutPutFile, ">>$File");
#
# print OutPutFile "$UserName,$AccessDate\n";
#
# close OutPutFile;
#}
#
#
#sub LAST_ACCESS {
#  local ($File, $Usrname) = @_;
#$LastAccess = "Never";
#
# open (Input, $File) | | die "Can't open $File: $!\n";
# while (<Input>) {
#   chop;
#   ($User, $Access) = split(/,/, $_);
#   if ($User eq $Usrname) {
#     $LastAccess = $Access;
#   }
# }
# close ($File);
# return $LastAccess;
#
#} # End of LAST_ACCESS


# ---------------
# Main Program
# ---------------

#These are markers in the data file for the beginning, commas, and end of text.

$Stuff{'Begin'} = "\xfb";
$Stuff{'End'} = "\xfd";
```

49

```perl
$Stuff{'Comma'} = "\xfc";
$Stuff{'BaseFile'} = '/home/dnet/DecisionNet/';

&ReadParse(*Stuff);
$Stuff{'Pwd'} = crypt($Stuff{'Pwd'}, Dnet);

$Stuff{'HeadNum'} = &WHERETOGO($Stuff{'UserType'});

$Stuff{'Headnum'} = 1;

if ($Stuff{'HeadNum'} eq 'Bogus' ) {   # Bogus Entry for User Type
  $Stuff{'Reject'} = 'Invalid Entry for User Type';
  &REJECT ($Stuff{'Reject'});
}
else {
  &CHECKFILE(*Stuff);  # Check for duplicate entries in file

  if ($Stuff{'Dup'} eq 'Yes') {  # Login and Password match
    # Send client the correct main menu -- Prov or User
    $AccessFile = $Stuff{'BaseFile'}.'data/'.$Stuff{'UserType'}.'.access.log';
    $LastAccess = &LAST_ACCESS($AccessFile,$Stuff{'Usrname'});
    &ACCESSLOG($AccessFile,$Stuff{'Usrname'});
    &OUTPUT(*Stuff, $LastAccess);
  }
  else {  # UserName or Password does not match
    $Reject = "Consumer Name or Password not Matching: ";
    &REJECT($Reject);
  }
}


__END__
```

APPENDIX E.   USER REGISTRATION

The User Registration is a common module for both Providers and Consumers. The user information is parsed by *cgi-lib.pl*. After parsing out the data, the passwords are checked for blank passwords. Assuming the passwords are not blank, the current user accounts are checked to verify that the username requested is not in use by another user of the same type -- Provider or Consumer. If this is a unique user name, then the passwords are checked to ensure that the same password was entered twice.

If all of this is completed with out an error, then the user entered data is split into three separate files -- *_Address.txt, *_Access.txt, and *_mail.txt. The user type variable is substituted for the *. All information is appended to a single file and stored in the clear. The single exception is the user password, which is stored in the encrypted form. Storing the encrypted form of the password ensures that the only time the password is sent across the Internet in the clear is during the initial registration of the account.

The account verification is sent back to the client via E-Mail -- one of the entered data fields. A short HTML document is generated and sent to the client informing them that the verification of the account will be sent via E-Mail.

```perl
#! /usr/bin/perl
#         Title: register.pl
#      Written by: Andrew King
#         Date: 25 Jan 95
#    Last Modified: 3 May 95
#         Version: 2.2
# Parsing by cgi-lib routines
#         Purpose: Capture the New Registration information into an
```

```perl
#                    encrypted file and put username/email into separate file


require 'dnetlib1.pl';
require 'cgi-lib.pl';


#-------------------------------
# Sub OUTFILE
# Output captured information to files
#-------------------------------

sub OUTFILE {
 local(*FORM) = @_;

# >> opens file in append mode
 $FORM{'File'} = $FORM{'BaseFile'}.'data/'.$FORM{'UserType'}.'_Address.txt';
 open (The_File, ">>$FORM{'File'}");

# Write out the address to THE_FILE

 print The_File "$FORM{'Begin'}$FORM{'Usrname'}$FORM{'Comma'}";
 print The_File "$FORM{'CompanyName'}$FORM{'Comma'}$FORM{'RealName'}";
 print The_File "$FORM{'Comma'}$FORM{'EMail'}$FORM{'Comma'}$FORM{'Street'}";
 print The_File "$FORM{'Comma'}$FORM{'City'}$FORM{'Comma'}$FORM{'State'}";
 print The_File "$FORM{'Comma'}$FORM{'Zip'}$FORM{'Comma'}";
 print The_File "$FORM{'Country'}$FORM{'End'}\n";
 close(The_File);

# Write out the access info to THE_FILE

 $FORM{'File'} = $FORM{'BaseFile'}.'data/'.$FORM{'UserType'}.'_Access.txt';
 open (FILE_2, ">>$FORM{'File'}");
 print FILE_2
"$FORM{'Begin'}$FORM{'Usrname'}$FORM{'Comma'}$FORM{'Pass'}$FORM{'End'}\n";
 close(FILE_2);

# Write out the email, and name to address to THE_FILE

 $FORM{'File'} = $FORM{'BaseFile'}.'data/'.$FORM{'UserType'}.'_mail';
 open(FILE_3, ">>$FORM{'File'}");
 print FILE_3 "$FORM{'Begin'}$FORM{'Usrname'}$FORM{'Comma'}$FORM{'Pwd'}";
 print FILE_3 "$FORM{'Comma'}$FORM{'EMail'}$FORM{'Comma'}";
 print FILE_3 "$FORM{'Access'}$FORM{'End'}\n";
 close(FILE_3);
```

```perl
}

#---------------------------------
# Output Subroutine
# Encrypts Data, send data to file, Sends verification to client via E-Mail
#---------------------------------

sub OUTPUT {  # Output to file and return verification to user
  local(*FORM) = @_;

# The Encryption Salt has been removed to ensure all passwords are secure
# Despite this thesis being released to the public.

  $FORM{'Pass'} = crypt($FORM{'Pwd'},SALT);

 # Get Time and Date for new registration Time
  $FORM{'Access'} = &DATETIME;
  &OUTFILE(*FORM);  # Output to the correct Files
  $Subject = 'DecisionNet Account Verification';

  $Body = "The following record has been added to the register as of $FORM{'Access'}\n
New User Name: $FORM{'Usrname'}\n \n Welcome to DecisionNet!\n";
  &EMAIL($FORM{'EMail'}, $FORM{'Usrname'}, $Subject, $Body);

#
}

sub RESULTS {
  local (*Input) = @_;
  $HeaderFile = $Input{'BaseFile'}.'Head/header1.head';
  $Title = "$Input{'Reject'}\n";
  &HEADER($HeaderFile, $Title, 'Header');

  print "<H2>Your confirmation will be sent via E-Mail to : $Input{'EMail'}</H2>";
  $Title = "";
  $OutputFile = $Input{'BaseFile'}.'Foot/footer1.foot';
  &HEADER($OutputFile, $Title, 'Footer');


} # End Reject

# ---------------------------------
# Subroutine Reject
# ---------------------------------
```

```perl
sub REJECT {
 local (*Input) = @_;

 $HeaderFile = $Input{'BaseFile'}.'Head/header1.head';
 $Title = "Registration Failure\n";
 &HEADER($HeaderFile, $Title, 'Header');


 print "<H3>New Account registration failed</H3>\n";
 print "Reason for failure: $Input{'Reject'}<br>\n";
 $Title = "";
 $OutputFile = $Input{'BaseFile'}.'Foot/footer1.foot';
 &HEADER($OutputFile, $Title, 'Footer');


} # End Reject




#--------------------------------
# Main Program
#--------------------------------

$Stuff{'Begin'} = "\xfb";
$Stuff{'End'} = "\xfd";
$Stuff{'Comma'} = "\xfc";
$Stuff{'BaseFile'} = '/home/dnet/DecisionNet/';

#Debugging line
# print "Content-type: text/html\n\n";

&ReadParse(*Stuff);

# Check for blank password or username before continue
if ($Stuff{'Usrname'} eq "") {
 $Stuff{'Reject'} = ' A blank Login Name.';
 &REJECT(*Stuff);
}
elsif ($Stuff{'Pwd'} eq "") {
 $Stuff{'Reject'} = 'A blank Password.';
 &REJECT(*Stuff);
}
else {
```

54

```perl
# Password and Username filled in
 &CHECKFILE(*Stuff);  # Check for duplicate entries in file

 if ($Stuff{'DupName'} eq 'Yes') {  # New Registration is a duplicate
  $Stuff{'Reject'} = 'Duplicate Registration.';
  &REJECT(*Stuff);
 }
 elsif ($Stuff{'Pwd'} eq $Stuff{'PassWord1'}) {  # Passwords match
  &OUTPUT(*Stuff);
  &RESULTS(*Stuff);
  }
 else {  # Passwords do not match
  $Stuff{'Reject'} = 'Passwords not matching.';
  &REJECT(*Stuff);
 }
}

__END__
```

APPENDIX F. TECHNOLOGY REGISTRATION

This function requires two Perl scripts -- *Register.Form.pl* and *register.pl*.
Register.Form.pl generates the registration form for the Provider using the
information entered at the time of registration to fill in the point of contact (POC)
information, and insert the hidden fields for the user-type, user name and
password. When the data for the technology is completed, the data is sent to
register.pl. *Register.pl* verifies the account information, then uses the data entered
to fill in the HTML form that is stored in a temporary directory. The provider is
sent an E-Mail to verify that DecisionNet has received the information and will
post it shortly.

The posting is currently done automatically via a *crontab* job at midnight.
The verification of the technology is done after registration is accepted.

```
#! /usr/bin/perl
#          Title: Register.Form.pl
#          Based on: add.pl and login.pl
#       Written by: Andrew King
#             Date: 9 Aug 95
#    Last Modified: 9 Aug 95
#          Version: 2.0
# Parsing by cgi-lib routines
#          Purpose: Capture the New Technology Registration information
#                   into a HTML file to be included in the search routine.


require 'dnetlib1.pl';
require 'cgi-lib.pl';

# -------------------------------
# Output UserType Menu
# -------------------------------
```

```perl
sub OUTPUT { # Output the New Registration Form

  local (*Input) = @_;
  &GetPOCInfo(*Input);

  $HeaderFile = $Input{'BaseFile'}.'Head/header1.head';

  $Title = "Register New Technology for ".$Input{'Usrname'}."\n";
# Output Header to Client

  &HEADER($HeaderFile, $Title, 'Header');

  $InputFile = $Input{'BaseFile'}.'Body/register.model.body';

# Calculate the age of the html file
  stat($InputFile);
  $Age = -M _;
  $Age = int($Age);

  open (File,$InputFile) | | die "Can't open $InputFile: $!\n";

  while (<File>) {

    chop;
    $Line = $_;
# Check for submit button to insert hidden fields

    if (/TYPE=submit/) { # Perhaps insert Hidden Fields

      print '<input name=UserType type=hidden value="',$Input{'UserType'},'">'."\n";
      print '<input name=Usrname type=hidden value="',$Input{'Usrname'},'">'."\n";
      print '<input name=Pwd type=hidden value="',$Input{'Pwd'},'">'."\n";
      print "$Line\n";
    }
    else { # Perhaps insert POC Info
      print "$Line\n";
      if (/POCName/) {
        print "$Input{'POCName'}";
      }
      elsif (/POCStreet/) {
        print "$Input{'POCStreet'}";
      }
      elsif (/POCCity/) {
        print "$Input{'POCCity'}";
      }
```

58

```perl
  elsif (/POCEMail/) {
   print "$Input{'POCEMail'}";
  }
  elsif (/POCCompany/) {
   print "$Input{'POCCompany'}";
  }
  elsif (/POCState/) {
   print "$Input{'POCState'}";
  }
  elsif (/POCZip/) {
   print "$Input{'POCZip'}";
  }
  elsif (/POCCountry/) {
   print "$Input{'POCCountry'}";
  }


  } # End else

 }   # End While loop

 close (InputFile);


print "<br><i>This file was last modified $Age Days ago</i><br>\n";

 $Title = "";
 $OutputFile = $Input{'BaseFile'}.'Foot/footer1.foot';
 &HEADER($OutputFile, $Title, 'Footer');

}   # End Output


# ----------------------------------
# Subroutine Reject
# ----------------------------------


sub REJECT {
 local (*Input) = @_;

 $HeaderFile = $Input{'BaseFile'}.'Head/header1.head';
 $Title = "Registration Failure\n";
 &HEADER($HeaderFile, $Title, 'Header');
```

```perl
    print "<H3>New Technology registration failed</H3>\n";
    print "Reason for failure: $Input{'Reject'}<br>\n";

    $Title = "";
    $OutputFile = $Input{'BaseFile'}.'Foot/footer1.foot';
    &HEADER($OutputFile, $Title, 'Footer');



}  # End Reject




#--------------------------------
# Main Program
#--------------------------------

$Stuff{'Begin'} = "\xfb";
$Stuff{'End'} = "\xfd";
$Stuff{'Comma'} = "\xfc";
$Stuff{'BaseFile'} = '/home/dnet/DecisionNet/';

# Debugging line  uncomment to start debugging
# print "Content-type: text/html\n\n";

&ReadParse(*Stuff);

if ($Stuff{'UserType'} ne 'Prov' ) {   # Bogus Entry for User Type
  $Stuff{'Reject'} = 'Your Provider Registration is not Valid. Please reregister with
DecisionNet prior to adding a new Technology.';
  &REJECT(*Stuff);
}
else {
  &CHECKFILE(*Stuff);  # Check for duplicate entries in file

  if ($Stuff{'Dup'} eq 'Yes') {  # Login and Password match
```

```perl
    # Send client the correct main menu -- Prov or User
    $AccessFile = $Stuff{'BaseFile'}.'data/'.$Stuff{'UserType'}.'.access.log';
    $LastAccess = &LAST_ACCESS($AccessFile,$Stuff{'Usrname'});
    &ACCESSLOG($AccessFile,$Stuff{'Usrname'});
    &OUTPUT(*Stuff);
  }
```

```perl
  else {  # UserName or Password does not match
    $Stuff{'Reject'} = 'You are not a registered Provider.  Please register as a Provider
prior to adding a new technology.';
    &REJECT(*Stuff);
  }
}


__END__    (Register.Form.pl)


#! /usr/bin/perl
#          Title: add.pl
#        Based on: register.pl
#      Written by: Andrew King
#            Date: 25 Jan 95
#    Last Modified: 20 July 95
#          Version: 2.2
# Parsing by cgi-lib routines
#          Purpose: Capture the New Technology Registration information
#                  into a HTML file to be included in the search routine.


require 'dnetlib1.pl';
require 'cgi-lib.pl';

#--------------------------------
# Sub OUTFILE
# Output captured information to files
#--------------------------------

sub OUTFILE {
  local(*FORM) = @_;

# >> opens file in append mode
  $FORM{'File'} = $FORM{'BaseFile'}.'Incoming/'.$FORM{'FileName'}.'.html';
  open (The_File, ">>$FORM{'File'}");

# Write out the address to THE_FILE

  print The_File '<h2>Run the <A href = "';
  print The_File "$FORM{'URL'}";
  print The_File '">';
  print The_File "$FORM{'Name'}</h2></A><br>\n";
```

```
print The_File '<A href ="#purpose">Purpose</a>, ';
print The_File '<A href ="#construction">Construction</a>, ';
print The_File '<A href ="#description">Description</a>, ';
print The_File '<A href ="#comments">Comments</a>';
print The_File "<br>\n";
print The_File "<hr noshade size = 10><p>\n";


print The_File "<B>Name:</B> $FORM{'Name'}<br>\n";
# print The_File "<B>Execution URL:</B> $FORM{'URL'}<br>\n";
 print The_File "<B>Modeling Paradigm:</B> $FORM{'paradigm'}<br>\n";
 print The_File "<B>Model Type: </B>$FORM{'ModelType'}<br>\n";
 print The_File "<B>Date Implemented: </B>$FORM{'Date'}<br>\n";
 print The_File "<B>Functional Area: </B> $FORM{'Area'}<br>\n";
 print The_File "<B>Domain: </B> $FORM{'Domain'}<br>\n";
 print The_File '<A Name="purpose">';
 print The_File "<b>Purpose:</a> </b> $FORM{'Purpose'}<p>\n";
 print The_File "<hr noshade size = 10><p>\n";

 print The_File "<B>Point Of Contact Name: </B> $FORM{'POCName'}<br>\n";
 print The_File "<B>Company Name: </B> $FORM{'POCCompany'}<br>\n";
 print The_File "<b>POC Street: </B> $FORM{'POCStreet'}<BR>\n";
 print The_File "<b>POC City: </B>$FORM{'POCCity'}<BR>\n";
 print The_File "<b>POC State/Province: </B>$FORM{'POCState'}<BR>\n";
 print The_File "<b>POC Postal Code: </B>$FORM{'POCZip'}<BR>\n";
 print The_File "<b>POC Country: </B>$FORM{'POCCountry'}<br>\n";
 print The_File "<b>POC Phone: </b>$FORM{'POCPhone'}<P>\n";
 print The_File "<hr noshade size = 10><p>\n";

print The_File '<A Name="description">';
print The_File "<B>Description:</a> </B>$FORM{'Descript'}<br>\n";
print The_File "<hr noshade size = 10><p>\n";

print The_File '<b><A NAME="construction"> Construction:</A>';
print The_File "</B>$FORM{'Const'}<br>\n";
print The_File "<B>Human Participation:</B>$FORM{'participation'}<p>\n";
print The_File "<B> INPUTS: </B>$FORM{'Inputs'}<br>\n";
print The_File "<B>OUTPUT: </B>$FORM{'Outputs'}<br>\n";
print The_File "<hr noshade size = 10><p>\n";

print The_File '<B><A NAME="hwsw">Hardware and Software of system executing';
print The_File " this technology:</A></B><p>\n";
print The_File "<B>Computer: </B>$FORM{'Computer'}<br>\n";
print The_File "<B>Storage: </B>$FORM{'Storage'}<br>\n";
```

```perl
   print The_File "<B>Peripherals: </B>$FORM{'Peri'}<br>\n";
   print The_File "<B>Language: </B>$FORM{'Language'}<br>\n";
   print The_File "<B>Documentation: </B>$FORM{'Docs'}<p>\n";
   print The_File "<b>Security Classification: </b>$FORM{'Class'}<br>\n";
   print The_File "<hr noshade size = 10><p>\n";

   print The_File '<H2><A NAME="comments">Comments:</A>';
   print The_File "</h2>$FORM{'Comments'}<br>\n";
   print The_File "<hr noshade size = 10><p>\n";

   print The_File "<b>Last Update:</b> $FORM{'Access'}<br>\n";


   close(The_File);

}

#----------------------------------
# Output Subroutine
# Writes data file, Sends notification to user via E-Mail
#----------------------------------

sub OUTPUT {  # Output to file and return verification to user
  local(*FORM) = @_;

# Create filename (hopefully unique) for new file
  $FORM{'FileName'} = $FORM{'Name'};

  $FORM{'FileName'} =~ y/a-zA-Z0-9//cd;

  # Get Time and Date for new registration Time
  $FORM{'Access'} = &DATETIME;

  &OUTFILE(*FORM);  # Output to the correct Files
  $Subject = 'New Technology Registration';

  $Body = "The following record has been added to the DecisionNet register as of
$FORM{'Access'}\n New Technology Name: $FORM{'Name'}\n  Added by:
$FORM{'Usrname'} \n";
  &EMAIL($FORM{'POCEMail'}, $FORM{'Usrname'}, $Subject, $Body);

#
}

# ----------------------------------
```

```perl
# Subroutine Reject
# --------------------------------


sub REJECT {
 local (*Input) = @_;

 $HeaderFile = $Input{'BaseFile'}.'Head/header1.head';
 $Title = "Registration Failure\n";
 &HEADER($HeaderFile, $Title, 'Header');


 print "<H3>New Technology registration failed</H3>\n";
 print "Reason for failure: $Input{'Reject'}<br>\n";

 $Title = "";
 $OutputFile = $Input{'BaseFile'}.'Foot/footer1.foot';
 &HEADER($OutputFile, $Title, 'Footer');


}  # End Reject


sub RESULTS {
 local (*Input) = @_;
 $HeaderFile = $Input{'BaseFile'}.'Head/header1.head';
 $Title = "$Input{'Reject'}\n";
 &HEADER($HeaderFile, $Title, 'Header');

 print "<H2>Your confirmation will be sent via E-Mail to : $Input{'POCEMail'}</H2>";
 $Title = "";
 $OutputFile = $Input{'BaseFile'}.'Foot/footer1.foot';
 &HEADER($OutputFile, $Title, 'Footer');


}  # End Results




#--------------------------------
# Main Program
#--------------------------------

$Stuff{'BaseFile'} = '/home/dnet/DecisionNet/';
```

64

```
#Debugging line
# print "Content-type: text/html\n\n";

&ReadParse(*Stuff);

# &CHECKFILE(*Stuff);  # Check for valid Provider Name

#if ($Stuff{'Dup'} eq 'Yes') {  # Provider Registered
  &OUTPUT(*Stuff);
  &RESULTS(*Stuff);
# }
# else {  # Provider Not Registered ... How did that happen?
#   $Stuff{'Reject'} = 'Your Provider Registration is not Valid. Please reregister with
DecisionNet prior to adding a new Technology.';
#   &REJECT(*Stuff);
# }


__END__
```

LIST OF REFERENCES

1.   Sprague, Ralph H. Jr., and Eric D. Carlson, *Building Effective Decision Support Systems*, (Englewood Cliffs, NJ: Prentice-Hall, Inc., 1982), 256-277 passim.

2.   Sprague, Ralph H. Jr., and Barbara C. McNurlin, *Information Systems Management in Practice*, Third Edition, (Englewood Cliffs, NJ: Prentice-Hall, Inc., 1993), 382-385.

3.   Chou, Hugh, *Mortgage Paymeny Query*, (*http://ibc.wustl.edu/mort.html*).

4.   webmaster@strategy.com, *MicroStrategy: Success*, (*http://www.strategy.com/msi_sin1.htm*).

5.   Ayre, Rick, and Don Willmott, "The Internet Means Business", *PC Magazine*, May 16, 1995, Vol. 14, No. 9, p200.

6.   www@www.merit.edu, *Merit Network, Inc.*, (*http://nic.merit.net*).

7.   *NSFNET Packet Traffic History*, (*gopher://nic.merit.edu:7043/00/nsfnet/statistics/history.packets*).

8.   Dern, Daniel P., *Internet Guide for New Users*, (New York: McGraw-Hill, Inc., 1994), 190.

9.   Bui, Tung X., ed., *Proceedings of the Third International Conference on Decision Support Systems*, (Hong Kong: Elsevier Publishing Co., 1995), Vol. I and II, 137-146, 499-528.

10. podonnel@fcit.monash.edu.au, *Decision Support Systems Research Group*, (*http://ponderosa.is.monash.edu.au/~podonnel/dss.html*).

11. webmaster@cis.ufl.ed, *UF/NA Perl Archive*, (*http://www.cis.ufl.edu/perl/*).

12. Denny, Robert B., *windows httpd*, (*http://www.city.net/win-httpd/*).

13.   Shotton, Chuck, *MacHTTP Technical Reference*,
      (http://www.biap.com/documentation/technical_ref.html).

14.   webmaster@lycos.com, *The Lycos Home Page: Hunting WWW Information*,
      (http://lycos.cs.cmu.edu/).

15.   Schwartz, Michael, and the Internet Research Task Force Research Group
      on Resource Discovery (IRTF-RD), *Harvest Information Discovery and Access
      System, (http://rd.cs.colorado.edu/)*.

16.   Hughes, Kevin Hughes (kevinh@eit.com), *SWISH Documentation*,
      (http://www.eit.com/software/swish/swish.html).

17.   Berners-Lee, Tim, Robert Cailliau, Ari Luotonen, Henrik Frystyk Nielsen,
      and Arthur Secret, "The World-Wide Web", *Communications of the ACM*,
      August 1994, vol. 37, No. 8, 82.

18.   MacArthur, Karen, *World Wide Web Initiative: The Project*,
      (http://www.w3.org/).

19.   Access log for sm.nps.navy.mil, parsed by Author.

20.   Luotonen, Ari, *Basic Protection Scheme for the WWW*,
      (http://www.w3.org/hypertext/WWW/AccessAuthorization/Basic.htm).

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center 2
   Cameron Station
   Alexandria, VA 22304-6145

2. Library, Code 52 2
   Naval Postgraduate School
   Monterey, CA 93943-5101

3. Professor Hemant Bhargava (Code SM/BH) 3
   Naval Postgraduate School
   Systems Management Department
   Monterey, CA 93943-5000

4. Professor Balasubramaniam Ramesh (Code SM/RA) 2
   Naval Postgraduate School
   Systems Management Department
   Monterey, CA 93943-5000

5. CAPT. Paul Bloch (Code OR/BC) 1
   Naval Postgraduate School
   Operations Research Department
   Monterey, CA 93943-5000

6. Professor Frank Petho (Code OR/PC) 1
   Naval Postgraduate School
   Operations Research Department
   Monterey, CA 93943-5000

7. CAPT. George Conner (Code OR/CO) 1
   Naval Postgraduate School
   Operations Research Department
   Monterey, CA 93943-5000

70